Laurent Valentin

Goto Hikaru

2ème année ENSEA

Projet de 2ème année à l'E.N.S.E.A.

Apprentissage Profond et Classification

Encadré par Michel Chapron

2017-2018

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

Remerciements

Nous tenons à remercier notre encadrant pour ce projet, Monsieur Chapron qui nous apporté une grande aide et de nous avoir consacré son temps. De même, nous tenons à remercier les encadrants d'option IA messieurs Vu et Luvizon pour les cours supplémentaires qui nous ont grandement aidé.

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

Table des matières

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

Introduction

L'année 2016 aura été celle des grandes percées en intelligence artificielle.

Mars Le programme AlphaGo de la filiale de Google "DeepMind" battait un champion coréen au jeu de go par quatre victoires à une.
Juin L'équipe chinoise du moteur de recherche Baidu annonçait des performances inégalées en traduction automatique.
Septembre Google réplique avec l'intégration de cette technique dans son célèbre outil de traduction.
Novembre Une équipe d'Oxford et de Google décrivait son programme de lecture sur les lèvres, surpassant nettement les meilleurs programmes, déjà supérieurs à l'être humain.

L'année 2016 a aussi été celle des assistants vocaux à la maison (Echo d'Amazon, Home de Google…), des « robots » de conversation, des véhicules autonomes.

Nous pouvons dire que l'avancée de l'apprentissage profond est grandement liée à l'avancée du Big Data, qui est de nos jours omniprésente. En effet, l'apprentissage profond, permet non seulement d'améliorer considérablement les résultats issus de programmation dites « classiques », mais surtout à résoudre des problèmes très complexes ou trop demandant en termes de ressources (calcul, temps) pour les humains.

Dès lors, il faut comprendre que la classification nécessite, d'une part une machine très puissante permettant de tourner des tâches très répétitives et très gourmandes, mais d'une autre part un algorithme prenant en compte toutes les conditions imaginables pour permettre de différencier un objet d'un autre. Or, en passant par le deep Learning, c'est maintenant possible, et cela, pour des cas simples comme distinguer un chat d'un chien, ne nécessitant, seulement qu'une dizaine de lignes de code.

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

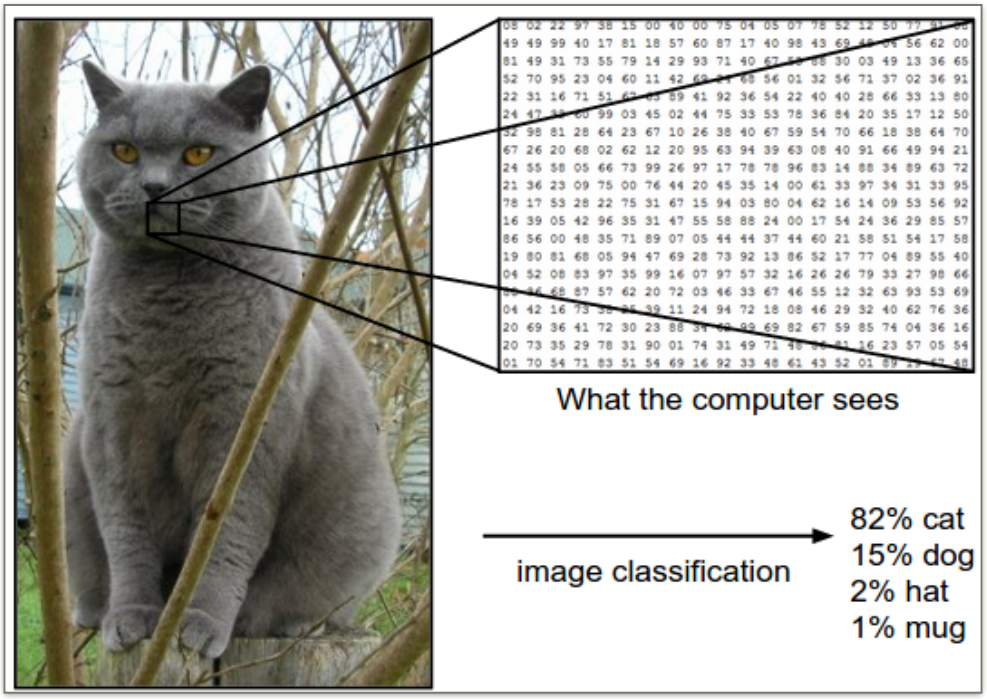
I. Pourquoi le Deep Learning

Dans cette partie, on mettra en évidence l'utilité du "Deep learning" pour implémenter un classificateur.

A. Complexité Homme Machine

2. Différents procédés

La machine ne perçoit pas une image comme un humain :



L'Homme perçoit ce qui l'observe à travers ses yeux, tandis que l'ordinateur reçoit des 0 ou des 1.
Pour un ordinateur, '' reconnaitre " un chat n'est pas évident.

2. Différents Apprentissages

Apprentissage Humain	Apprentissage d'une machine
Par répétition, expérience	Il n'existe pas d'apprentissage en programmation classique

I. Pourquoi le Deep Learning

Une programmation classique consiste à écrire des fonctions qui modélisent le fonctionnement du programme, avec les conditions et les contraintes que l'on définit.

Cela revient à lui faire apprendre une tâche à effectuer, et la machine se contentera d'exécuter la tâche. Ainsi, il n'y a pas d'apprentissage par lui-même à proprement parlé.

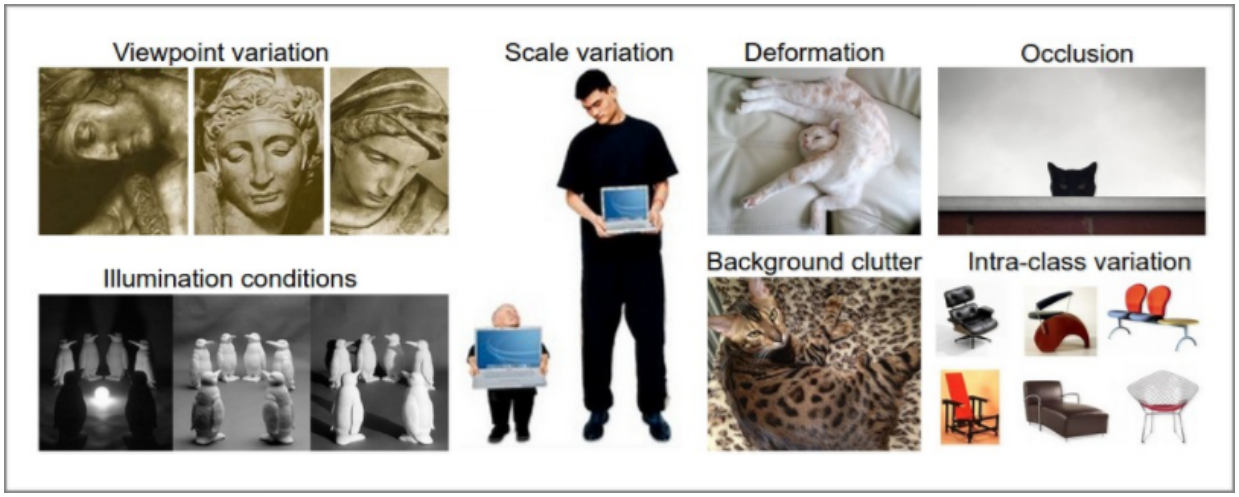
B. Classification

1. Classification par conditionnement

La classification d'une image ne peut pas être basée sur une programmation à conditionnement, car l'entrée est d'une forme trop complexe :

Il faudrait alors faire ressortir les caractéristiques d'un objet à travers des pixels, or ces caractéristiques sont très variantes par le bruit, l'angle de prise de vue, l'occlusion ou encore la déformation de l'objet.

En voici quelques-uns :



Il est donc très difficile pour nous de trouver les conditions nécessaires à l'ordinateur pour qu'il classifie correctement un objet d'une photo. Il faut trouver un autre moyen de classifier les images.

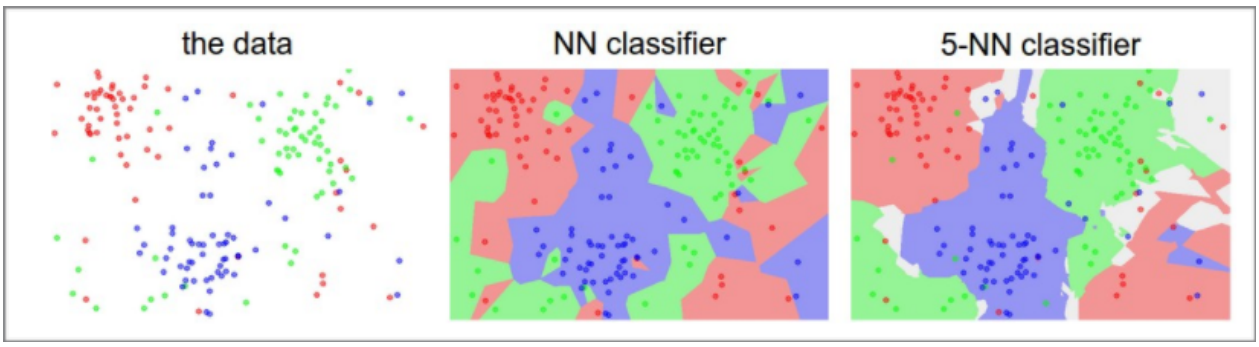
I. Pourquoi le Deep Learning

2. *KNN: K-Nearest Neighbors*

Ce classificateur se base sur une classification par ressemblance :

Il a en sa possession une base d'images classifiées. Il compare la différence en termes de valeurs de pixels entre l'image que l'on veut classer avec chacune des images de sa bade d'images, pour en trouver les k plus proche voisins. Il pourra ensuite classer cette image dans la catégorie d'images des plus proches voisins.

Voici un exemple :



On remarque que le nombre de voisin à prendre en compte influe beaucoup sur la forme de la classification, que l'on devra déterminer. De même, il faut déterminer la manière de calculer la ressemblance, entre la norme L1, L2 ou autres. Ces paramètres sont appelés Hyperparamètres et leur détermination devient clé : Or, cette détermination n'est pas toujours facile. Il faut les tester un par un.

Limite du KNN:

- Très gourmand en ressources : pour chaque détermination, le classificateur a besoin de balayer toutes les images de sa base de données.

- Très gourmand en mémoire : ce classificateur présuppose une base d'images connue, ainsi il doit apprendre toutes les images de sa base de données pour effectuer l'analyse d'une nouvelle image.

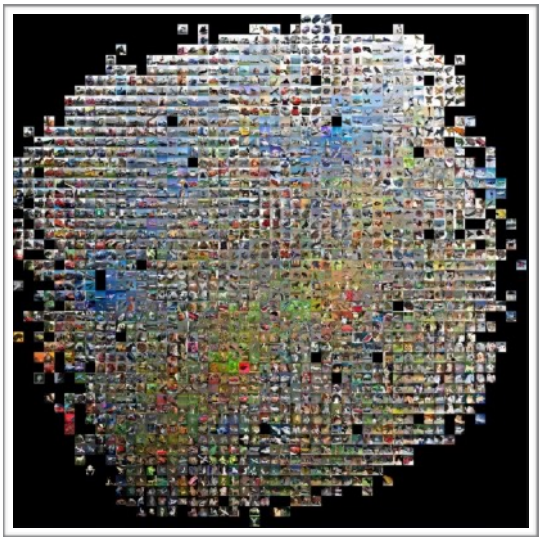
- Comparaison se basant sur la couleur du fond plutôt que de l'image considérée comme on peut le voir sur la page suivante.

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

I. Pourquoi le Deep Learning

Voici les images classées par un KNN, comportant des images de 10 classes différentes classées. On constate que la classification a été faite plus sur la teinte de l'image, que du contenu.

Ceci est dû à la comparaison directe de pixels, et non pas de figures/formes présentes dans une image.



Une classification d'images nécessite une analyse plus profonde qu'une simple comparaison de pixels, ce qui nous amène à une classification par Deep Learning.

II. Les principes du Deep Learning

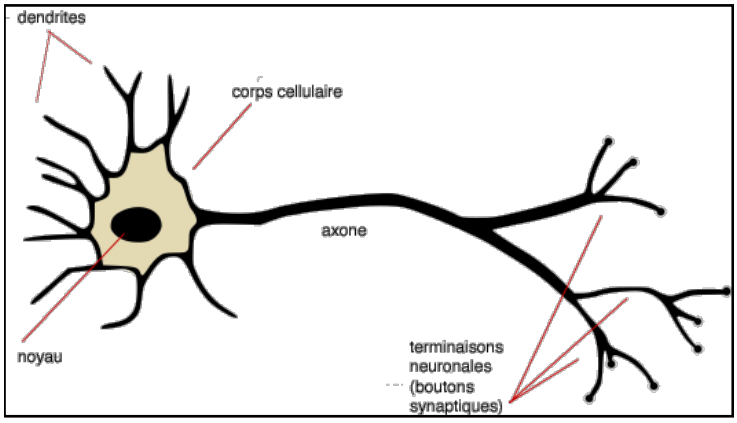
Le “Deep Learning“ (apprentissage profond) est, d’après Wikipédia, un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires. Ainsi, de par cette caractéristique, il permet de résoudre divers problèmes tels que les NP-difficiles.

1. *Fonctionnement d’un neurone formel*

L’origine du Deep Learning est issue du fonctionnement du neurone humain qui reçoit l’information grâce à ses multiples dendrites et la transmet le long d’un unique axone via les terminaisons nerveuse comme on peut le voir sur le schéma ci-dessous:

Principe du neurone humain: source de transmission de l’information

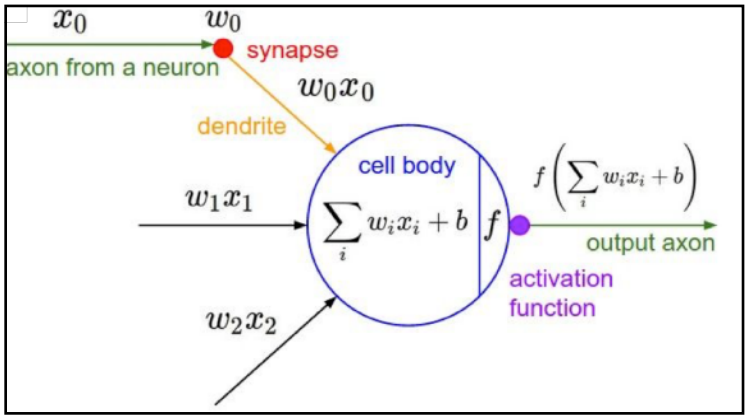
Ainsi de manière analogue au neurone humain, on utilise un modèle de neurone formel



qui a la structure suivante:

Modèle du neurone formel

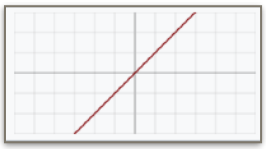
II. Les principes du Deep Learning



On peut donc voir que chaque neurone formel reçoit plusieurs entrées $x_1, x_2, \ldots x_n$ lesquelles sont multipliées par ce qu'on appelle des "poids" qui correspondent aux coefficients $w_1, w_2, \ldots w_n$. Par la suite, on effectue la somme de chacun des termes à laquelle on rajoute un bias noté b. Enfin la sortie de chaque neurone formel correspond au résultat de $f(\sum x_i w_i)$ où f est ce qu'on appelle la "fonction d'activation".

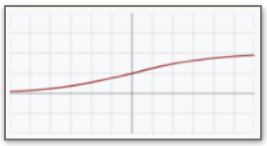
Il existe différents types de fonction d'activation:

-Identité $f(x) = x$



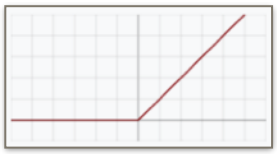
-Heavyside $f(x) = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$

-Sigmoid $f(x) = \dfrac{1}{1 + e^{-x}}$

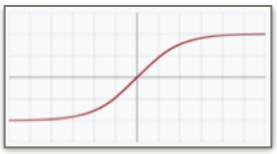


-ReLU $f(x) = \begin{cases} x & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$

(Unité de Rectification Linéaire)



-Tanh $f(x) = tanh(x)$



II. Les principes du Deep Learning

Ainsi si la sortie d'un neurone formel est égale à 0, cela revient à ne pas considérer l'entrée que celui-ci a reçu.

On définit alors un réseau de neurones comme un ensemble de neurones organisés en couche. Le réseau de neurone le plus simple est celui du perceptron, il s'agit d'un classificateur linéaire monocouche dans sa version la plus simple.

2. *Phénomène d'Apprentissage*

Tout réseau de neurone suit alors un algorithme d'apprentissage qui suit les étapes suivantes:

Algorithme d'apprentissage:
1. Initialisation des poids initiaux
2. Présentation au réseau de neurone d'un exemple
3. Calcul de la valeur en sortie du réseau
4. Mise à jour des poids en fonction des entrées/sorties
5. Retour à l'étape 2 jusqu'à ce que la précision désirée soit atteinte

Lorsqu'il y a activation il y a alors augmentation du poids associé w_i , sinon on diminue ce poids.

On définit alors plusieurs types d'apprentissage:

-L'apprentissage supervisé, selon lequel on fournit au réseau la cible à prédire:

$$D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)\}$$

-L'apprentissage non supervisé, selon lequel on ne fournit pas la cible au réseau:

$$D = \{x_1, x_2, \ldots, x_n\}$$

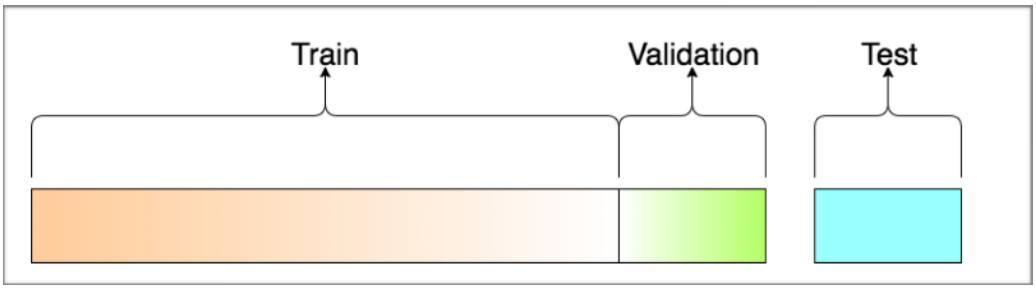
-L'apprentissage par renforcement, non traité dans ce compte-rendu.

La méthode d'apprentissage qui sera retenue sera celle de l'apprentissage supervisée.

II. Les principes du Deep Learning

3. *Organisation des données pour l'apprentissage du réseau de neurones*

Dans le but d'évaluer la capacité de notre modèle, on découpe notre base de données en trois parties: un "*ensemble d'entrainement*", un "*ensemble de test*" ainsi qu'un "*ensemble de test*".



Splitting of the data in three sets

Ainsi, on choisit généralement un ensemble d'entrainement équivalent à 80% des données d'entrainement et donc un ensemble de validation de 20%. Enfin, l'ensemble de test correspond à un ensemble de nouveaux échantillons que notre réseau n'aura jamais rencontré.

4. *Evaluation du modèle à l'aide d'une fonction de perte*

De plus, chaque réseau de neurones est caractérisé par ce qu'on appelle une "fonction de perte", elle permet de caractériser l'erreur effectuée à chaque exemple présenté au réseau. Une des fonctions de perte utilisée est celle de la distance:

$$L(w) = \frac{1}{2}\sum_{k=1}^{n}\{y(x_k, w) - t_k\}^2$$

Ainsi, lors de l'algorithme d'apprentissage supervisé, le réseau de neurone va chercher à minimiser cette fonction de perte $L(w)$ qui dépend des différents poids $w_1, w_2, \ldots w_n$, ce qui revient donc à chercher le vecteur W tel que $\frac{dL(W)}{dW} = 0$.
Ceci est réalisé grâce à la méthode de "Descente du Gradient" qui peut être stochastique ou par mini-lots. Nous avons choisi, lors de l'implémentation de notre réseau la stochastique.

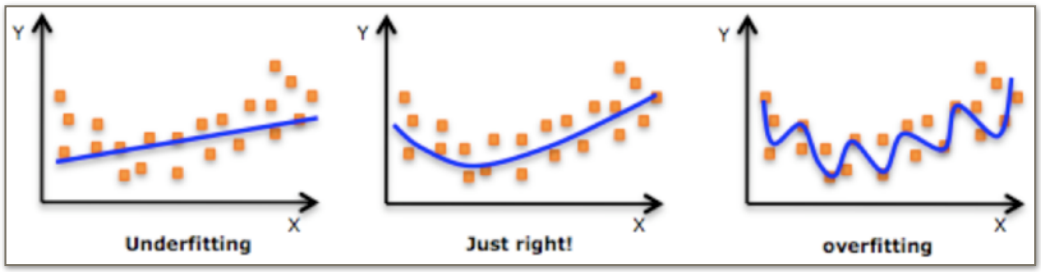
II. Les principes du Deep Learning

5. *Phénomènes d'Underfitting et d'Overfitting*

Cependant, lors du phénomène d'apprentissage, notre algorithme est susceptible de ne pas rencontrer assez d'exemples afin d'établir un modèle aussi complexe que le modèle étudié, c'est
ce qu'on appelle le "sous-apprentissage". Afin de contrer ce phénomène, il suffira de choisir un grand nombre d'échantillons dans notre base de données.
Ceci permet alors de comprendre pourquoi l'utilisation du Deep Learning est très en vogue dans la mesure où nous vivons l'émergence du Big Data.

De manière analogue, si nous montrons à notre modèle un trop grand nombre d'échantillons, ce dernier sera alors "trop complexe" dans la mesure où il saura reconnaitre les différents exemples qui lui ont été présentés précédemment mais n'aura alors aucune capacité à généraliser à de nouveaux échantillons. C'est ce que l'on appelle le "sur-apprentissage".

Nous pouvons résumer ces deux phénomènes à l'aide du schéma suivant:



C'est pourquoi, pour contrer l'"Overfitting", on introduit un terme de régularisation, noté λ, qui va permettre de prendre en compte, lors de l'évaluation de notre modèle, la complexité de ce dernier.

On aura alors une fonction de perte sous la forme:

$$L(w) = \frac{1}{2}\sum_{k=1}^{n}\{y(x_k, w) - t_k\}^2 + \frac{\lambda}{2} \parallel W \parallel^2$$

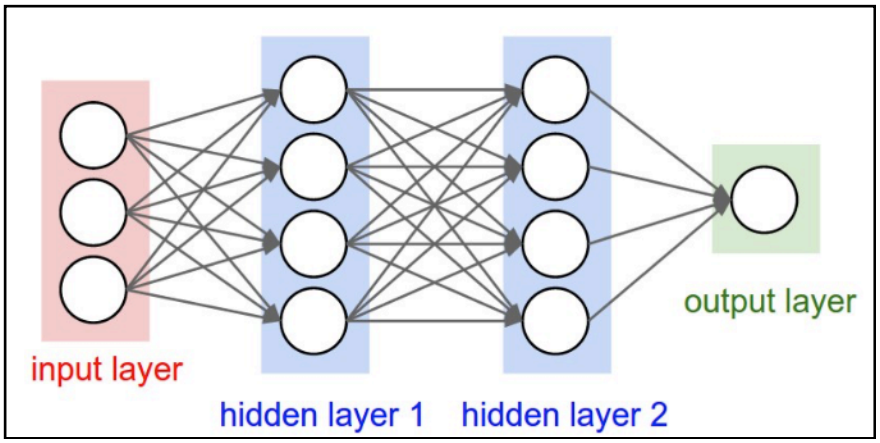
II. Les principes du Deep Learning

6. *Hyperparamètres*

On appelle “hyperparamètres“ le paramètre λ ainsi que l’ordre du modèle noté M.
Grâce à l’ensemble de validation, vu précédemment, il sera alors possible de comparer les résultats obtenus à l’aide de différents modèles du point de vue de leur capacité d’apprentissage “par coeur“ mais aussi leur capacité de généralisation.
Une fois les hyperparamètres choisis, on entraine notre modèle avec l’ensemble d’entrainement entier. L’ensemble de test nous permettra alors d’évaluer la taux de réussite de notre modèle et de continuer à le ré-entrainer si la précision souhaitée n’est pas atteinte.

7. *Structure classique d’un réseau de neurones*

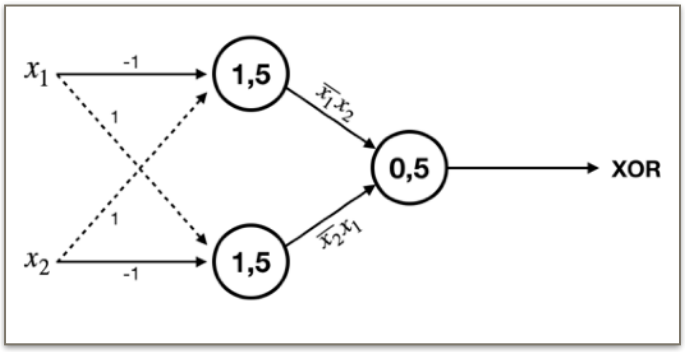
On peut définir la structure d’un réseau de neurones à l’aide du schéma suivant:



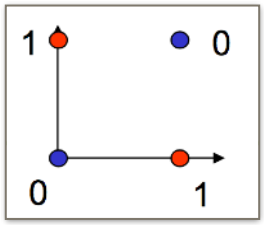
On peut donc voir la présence d’une couche d’entrée et une de sortie ainsi que ce que l’on appelle des “couches cachées“. Celles-ci sont dîtes cachées dans la mesure où nous n’avons pas directement accès à celles-ci afin de modifier leurs paramètres.

Pourquoi est-il nécessaire d’avoir un réseau a plusieurs couches ?
Ceci est dû au fait qu’il est nécessaire que notre réseau de neurones comporte une part de non-linéarité afin de réaliser une des fonctions de base les plus utilisées: la fonction “*XOR“ (“OU EXCLUSIF“). Cette dernière peut se modéliser à l’aide du réseau suivant:*

Modélisation d’un XOR à l’aide d’un réseau de neurones à 2 couches



Représentation du XOR



II. Les principes du Deep Learning

Voici différents types de couches qui sont fréquemment utilisées

-*Dense Layer* (couche fully connected)
-*Convolution Layer* (couche de convolution à l'aide d'un filtre pouvant être 2D pour le traitement d'images)
-*Pooling Layer* (couche permettant d'effectuer un échantillonnage des données)
-*Drop Layer* (couche permettant de ne prendre en compte qu'un pourcentage des données, elle permet de contrer le phénomène d'Overfitting)

8. *Rétropropagation du Gradient*

Le principe de la rétropropagation du gradient ("BackPropagation") est de calculer le gradient de l'erreur pour chaque neurone constituant chaque couche du réseau de neurones utilisé. Ce dernier est basé sur ce que l'on appelle la "chain rule" suivant laquelle on peut écrire:

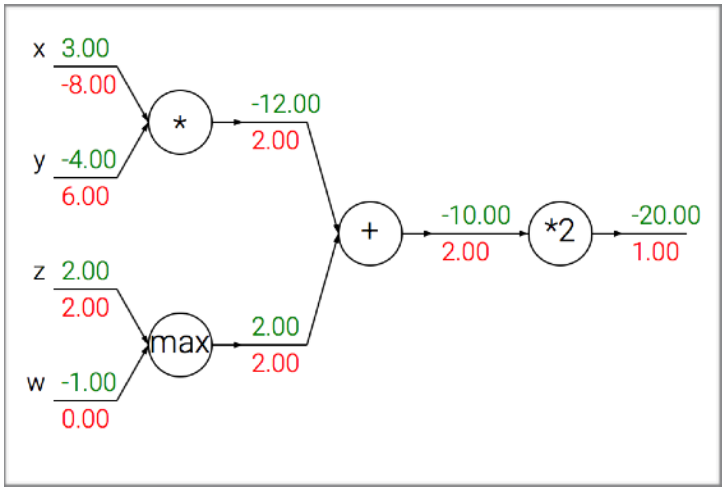
$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}u} \cdot \frac{\mathrm{d}u}{\mathrm{d}x}$$

Ainsi, on peut étudié l'importance de chaque poids dans le réseau et en déduire l'impact qu'il a sur le résultat de la fonction de perte. On peut illustrer ce principe avec le schéma ci-dessous.

Considérons la fonction suivante: $f(x, y, z, w) = 2[xy + max(z, w)]$
En supposant qu'on ait :

$$L(f) = -20 \quad x = -8 \quad y = 6 \quad z = 2 \quad w = 0$$

Graphe Computationnel de la fonction f



II. Les principes du Deep Learning

on peut en déduire le gradient de f :

$\frac{df}{dx}=3$ $\frac{df}{dy}=-4$ $\frac{df}{dz}=2$ et $\frac{df}{dw}=-1$

III. Implémentation d'un réseau de neurones

A. Mise en place d'un environnement de programmation

Pour pouvoir coder notre réseau de neurones, nous avons dû choisir un environnement de programmation.

L'objectif de notre projet étant d'évaluer la performance d'une programmation en Deep Learning, une programmation simple et rapide était primordiale.

En effectuant des recherches, nous avons remarqué que Python disposait de diverses bibliothéques destinées à la construction de réseaux de neurones, tels que Tensorflow et Theano.
Nous avons choisi de prendre la bibliothéque la plus récente qui était Tensorflow. De plus, nous avons utilisé Keras pour une syntaxe encore plus simple que celle de Tensorflow, ce qui nous a permis de nous concentrer sur le fond que sur la forme de la programmation.

Voici quelques étapes à suivre pour pouvoir coder des réseaux de neurones sur Python :

1. *Installer Python*

   - Utiliser la distribution Anaconda (Windows/Mac)

Lien : https://conda.io/docs/user-guide/install/index.html

Cela permet de gérer facilement les packages nécessaires pour le bon fonctionnement de Python.
En effet, python nécessite diverses packages pour bien fonctionner, comme par exemple Numpy pour travailler avec des matrices, ou encore Matplotlib pour tracer des courbes.
Grâce à la distribution Anaconda, l'installation de ces packages devient très facile, d'autant plus qu'elle se chargera des incompatibilités par rapport aux versions de Python.

Pour notre part, nous avons utilisé anaconda, puis installer Spyder/Jupyter dessus.

III. Implémentation d'un réseau de neurones

- Installation directe (Linux)

A l'aide du terminal, il suffit de taper :

```
sudo apt-get install python3.6
```

pour Python 3, et

```
sudo apt-get install python2.7
```

pour Python 2

2. *Installer Tensorflow*

a) Qu'est-ce que c'est ?

Tensorflow est un Framework (une bibliothéque qui présuppose un certain codage permettant de fonder les grandes lignes du programme) destiné au Deep Learning.
En passant par Tensorflow, il nous sera plus nécessaire de programmer toutes les fonctions utilisées dans un réseau de neurones, (fonctions d'activations, de régularisations etc.).

b) Installation

L'installation de Tensorflow passe par le terminal :

Ouvrir un terminal conda puis (Windows, linux):

```
anaconda create -n tensorflow pip python= votre_version_du_python(3.5 par exemple)

activate tensorflow

pip install --ignore-installed --upgrade tensorflow
```

III. Implémentation d'un réseau de neurones

Ouvrir un terminal de Linux, puis:

```
pip3 install --upgrade tensorflow
```

pour Python 3, et

```
pip install --upgrade tensorflow
```

pour Python 2.

4. *Installer Keras*

a)A quoi ça sert ?

Le passage par Keras permet de simplifier considérablement le codage du réseau de neurones, avec une syntaxe facile à lire et à comprendre.

b)*Installation*

L'installation de Keras passe aussi par le terminal. Pour Windows/Mac/Linux:

Ouvrir un terminal (conda pour windows et mac), puis :

```
sudo pip install keras
```

B. Programmation

1. *Librairies*

Il faut importer plusieurs librairies de Keras pour créer un réseau de neurones :

En voici quelques-uns que nous avons utilisé avec leur explication :

- keras.models

Contient 2 types de modèles de réseaux que peut créer Keras : Sequential et Model class with API
On utilisera que le Sequential, voir ici pour plus d'informations sur les 2 modèles.

- keras.layers

III. Implémentation d'un réseau de neurones

Contient les différentes couches du réseau. Les principaux sont :

- Dense ; un fully connected layer
- Conv2D ; une couche de convolution
- MaxPooling2D ; un pooling pour les conv2D
- Flatten ; vectorisation d'une matrice (nécessaire pour passer d'un Conv2D à un Dense)

- Keras.preprocessing

Permet de créer des batches de tenseur d'images, permettant des configurations aisées sur les images. (Voir ici pour plus d'informations.)

2. *Structure du code*

Voici les grandes lignes d'une programmation sur Keras :

1. Déclaration d'un réseau

```
# Initialising the CNN
classifier = Sequential()
```

2. Création des layers

```
#1st Layer
classifier.add(Conv2D(64, (3, 3), input_shape = (64, 64, 3), strides=(1,1) ,activation = 'relu'))


# 2nd Layer
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))

# 3rd Layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))


# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 3, activation = 'softmax'))
```

III. Implémentation d'un réseau de neurones

3. Compilation des layers en un réseau :

```
# Compiling the CNN
classifier.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

4. Recalibrage des images (datagenerator pour utiliser le GPU)

```
# Part 2 - Fitting the CNN to the images



train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data2/train',
                                                 target_size = (64, 64),
                                                 batch_size = 64,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data2/validation',
                                            target_size = (64, 64),
                                            batch_size = 64,
                                            class_mode = 'categorical')
```

5. Exécution du réseau :

```
classifier.fit_generator(training_set,
                         steps_per_epoch = 2000,
                         epochs = 10,
                         validation_data = test_set,
                         validation_steps = 1500)
```

III. Implémentation d'un réseau de neurones

6. Sauvegarde/chargement des poids d'un réseau :

```
#Sauvegarde des poids W du modèle une fois celui-ci entrainé
classifier.save_weights('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/testSoftmax.h5')

#Chargement des poids W du modèle entrainé
classifier.load_weights('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/testSoftmax.h5')
```

7. Afficher la structure du réseau/ tester le programme

```
#Affichage de la structure du modèle
print("\n Voici la structure de votre réseau de neurones :")
classifier.summary()


#Evaluation du modèle sur un échantillon donné
s=next(training_set)
score=classifier.evaluate(s[0],s[1])
score=score[1]*100
print("Le taux de reussite de ce modele est de : 0,%d " % score )
#Demande une entrée à l'utilisateur
addr=raw_input("Entrez le nom de l'image à analyser \n")
addr="/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data2/test/"+addr

#new_img=image.load_img('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data/test/test.jpg')
#new_img_resized=image.load_img('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data/test/test.jpg',target_size=(64,64)



new_img=image.load_img(addr)
new_img_resized=image.load_img(addr,target_size=(64,64))

#Affichage du Résultat sur une nouvelle image présente dans un dossier

print("\n Voici l'image que vous souhaitez analyser: ")
plt.imshow(new_img)
plt.show()
new_data = np.array(new_img_resized)
new_data=np.expand_dims(new_data, axis=0)

prediction=classifier.predict(new_data,batch_size=None, verbose=0)
prediction=prediction[0].tolist()
s=prediction.index(max(prediction))

if s==0:
    print"Il s'agit d'une voiture"
elif s==1:
    print"Il s'agit d'un chat"
else:
    print"Il s'agit d'un chien"
```

III. Implémentation d'un réseau de neurones

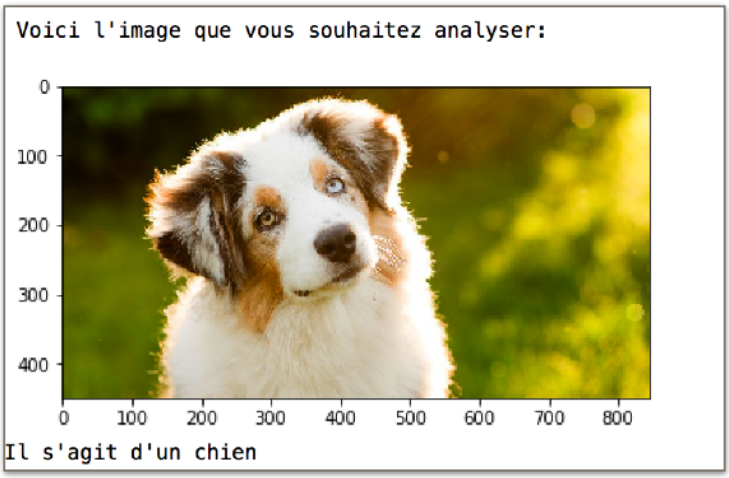
3. Résultats

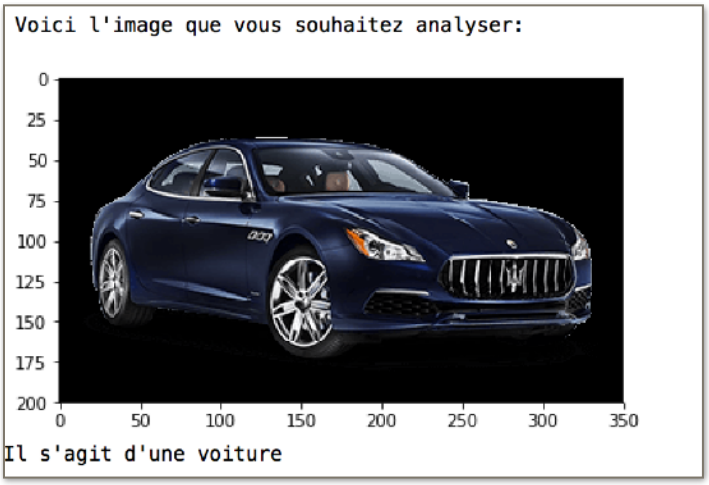
Voici la structure finale de notre réseau :

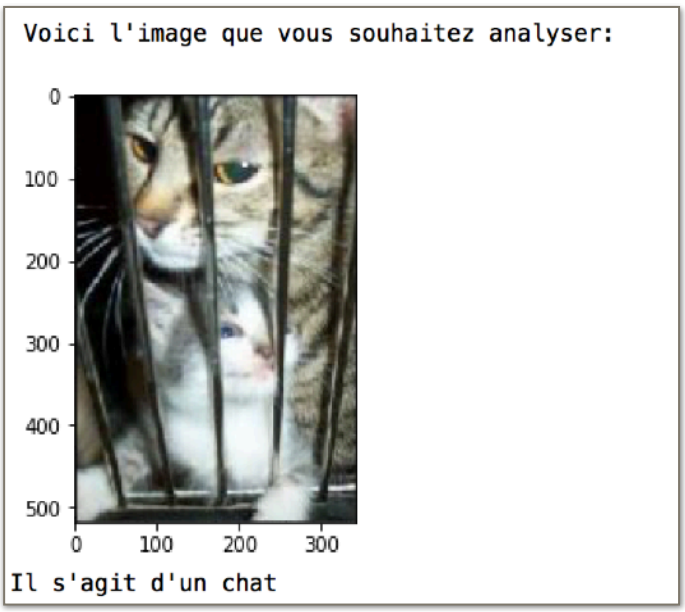
```
 Voici la structure de votre réseau de neurones :
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 62, 62, 64)        1792
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 58, 58, 32)        18464
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 29, 29, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 26912)             0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               6889728
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 387
=================================================================
Total params: 6,980,195
Trainable params: 6,980,195
Non-trainable params: 0
_________________________________________________________________
64/64 [==============================] - 1s 11ms/step
Le taux de reussite de ce modele est de : 0,93
```

Nous avons implémenter un réseau de neurones capable de différencier trois classes: “chien“, “chat“ et “voiture“. On peut remarquer que, pour cela, on a donc implémenté un réseau de neurones constitué de trois couches de convolution 2D, une couche d'échantillonnage, une couche Flatten ainsi que 3 couches Fully connected. Par ailleurs, le pourcentage théorique de réussite en moyenne est de 93%. Afin d'obtenir ce résultat assez élevé nous avons utilisé des base de données de 25000 images de chiens et de chats ainsi que plus de 20000 images de voitures et la durée de la phase d'apprentissage est de l'ordre d'une dizaine d'heures.

Voici des exemples de ce qu'est capable de reconnaitre notre réseau de neurones:







Bibliographie

- Nombreux cours sur les réseaux de neurones à convolution de l'université de Stanford disponibles sur le site http://cs231n.stanford.edu

- Tutoriel afin d'implementer un classificateur entre chiens et chats sur le site https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

- Guide d'installation et de prise en main de la bibliothèque Tensorflow à l'adresse https://www.tensorflow.org/install/install_mac

- Explications sur l'implémentation de la structure d'un réseau de neurones sur le site https://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/

- Articles expliquant les avancées du "Deep Learning" dans le magazine Le Monde http://www.lemonde.fr/sciences/article/2017/01/09/la-revolution-des-neurones-artificiels_5059943_1650684.html

- Livres Deep Learning avec Tensorflow et Machine Learning avec SciKit-Learn écris par Aurélien Géron

- Cours de M. Luvizon, disponibles sur son site http://perso-etis.ensea.fr/luvizon/pluxml/

- Manipulation de bibliothèques pour le traitement d'images à l'adresse https://www.normalesup.org/~dconduche/prepas/PTSI/TP/FeuilleImages_2013.pdf

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

Annexe n°1:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  3 14:59:28 2018

@author: Valentin LAURENT and Hikaru GOTO
"""

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.preprocessing import image
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import glob
import os.path

# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution

#1st Layer
classifier.add(Conv2D(64, (3, 3), input_shape = (64, 64, 3), strides=(1,1) ,activation = 'relu'))


# 2nd Layer
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))

# 3rd Layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

```
# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 3, activation = 'softmax'))

# Compiling the CNN
classifier.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics = ['accuracy'])

# Part 2 - Fitting the CNN to the images




train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data2/train',
                                                 target_size = (64, 64),
                                                 batch_size = 64,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data2/validation',
                                            target_size = (64, 64),
                                            batch_size = 64,
                                            class_mode = 'categorical')



#classifier.fit_generator(training_set,
#                         steps_per_epoch = 2000,
#                         epochs = 10,
#                         validation_data = test_set,
#                         validation_steps = 1500)
#
```

Compte-Rendu Projet 2ème année
Apprentissage Profond et Classification

(Suite) Annexe n°1

```
#Sauvegarde des poids W du modèle une fois celui-ci entrainé
classifier.save_weights('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/testSoftmax.h5')

#Chargement des poids W du modèle entrainé
classifier.load_weights('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/testSoftmax.h5')

#Affichage de la structure du modèle
print("\n Voici la structure de votre réseau de neurones :")
classifier.summary()


#Evaluation du modèle sur un échantillon donné
s=next(training_set)
score=classifier.evaluate(s[0],s[1])
score=score[1]*100
print("Le taux de reussite de ce modele est de : 0,%d " % score )
#Demande une entrée à l'utilisateur
addr=raw_input("Entrez le nom de l'image à analyser \n")
addr="/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data2/test/"+addr

#new_img=image.load_img('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data/test/test.jpg')
#new_img_resized=image.load_img('/Users/Valentin/Desktop/Artificial_Intelligence/ImageClassification/data/test/test.jpg',target_size=(64,64)



new_img=image.load_img(addr)
new_img_resized=image.load_img(addr,target_size=(64,64))


#Affichage du Résultat sur une nouvelle image présente dans un dossier

print("\n Voici l'image que vous souhaitez analyser: ")
plt.imshow(new_img)
plt.show()
new_data = np.array(new_img_resized)
new_data=np.expand_dims(new_data, axis=0)

prediction=classifier.predict(new_data,batch_size=None, verbose=0)
prediction=prediction[0].tolist()
s=prediction.index(max(prediction))

if s==0:
    print"Il s'agit d'une voiture"
elif s==1:
    print"Il s'agit d'un chat"
else:
    print"Il s'agit d'un chien"


#print(prediction)
#
#print("C'est un chat avec une probablité de : %s " %(prediction[0][1]))
#print("C'est un chien avec une probablité de : %s " %(prediction[0][2]))
#print("C'est une voiture avec une probablité de : %s " %(prediction[0][0]))
```